Noah Buchanan
Problem Set 4
Distributed Systems

October 27, 2020

File Management system from problem set 1, none of the computational complexity has been increased or otherwise changed, it simply supports threads now in a safe environment. Locking any resources deemed necessary and supports read and write locks on UAFile's for appending to files and reading files.

LionsFSManager Class

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.concurrent.locks.ReentrantLock;

/*******************************
Name: Noah Buchanan
Username: dist103
Problem Set: PS4
Due Date: October 27, 2020
*******************************/


/**
 * File management system utilizing all the appropriate methods needed for su
 * activities supports threading
 *
 * @author noah_
 *
 */
```

```
public class LionsFSManager implements Runnable {

  public HashMap<String, UAFile> Files = new HashMap<>();
  public HashMap<String, UACategory> Categories = new HashMap<>();
  int task = 1;
  final ReentrantLock lock = new ReentrantLock();
  boolean locktimer = true;

  /**
   *
   * @param filename name of file to lock
   * @param write    boolean determining whether the lock requires a readLock
   *                 writeLock
   * @return stamp for the specified lock
   * @throws UAInvalidFileException
   */

  public long lock(String filename, boolean write) throws UAInvalidFileExcept

    if (write) {

      synchronized (Files) {

        return Files.get(filename).lock.writeLock();
      }
    } else {

      return Files.get(filename).lock.readLock();
    }

  }

  /**
   *
   * @param filename name of file to unlock
   * @param stamp    stamp required to unlock the correct lock
   * @param write    boolean determining whether the lock to be unlocked is
   *                 readLock or writeLock
   * @throws UAInvalidFileException
   */

  public void unlock(String filename, long stamp, boolean write) throws UAIn

    synchronized (Files) {

      Files.get(filename).lock.unlock(stamp);
```

```
  }
}

/**
 * implements a readLock() on each file as it loops through their contents
 *
 * @param category Category of files, whom contents we will be listing
 * @return simply returns a concatenated string of the contents of all files
 *         listed
 * @throws IOException
 * @throws UAInvalidCategoryException
 * @throws UAInvalidFileException
 */

public String printAllFiles(String category)
    throws IOException, UAInvalidCategoryException, UAInvalidFileException

  String concat = "";

  synchronized (Categories) {

    for (UAFile j : Categories.get(category).getAssociatedFiles().values())

      long stamp = lock(j.getFileName(), false);

      BufferedReader br = new BufferedReader(new FileReader(j.getPathToFile

      String line = "";

      while ((line = br.readLine()) != null) {
        System.out.println(j.getFileName() + "'s contents---->" + line);
        concat += j.getFileName() + "'s contents---->" + line + "\n";

      }

      unlock(j.getFileName(), stamp, false);

      br.close();

    }
  }

  return concat;
}

/**
```

```
 * implements a writeLock on the files as it loops through files to append
 *
 * @param category        Category of files, whom we will be appending a giv
 *                        String to
 * @param contentToAppend the content of which we will be appending to each
 *                        in the given category
 * @return returns previous content of files with the new content appended
 * @throws IOException
 * @throws UAInvalidCategoryException
 * @throws UAInvalidFileException
 */

public String appendToEachFile(String category, String contentToAppend)
    throws IOException, UAInvalidCategoryException, UAInvalidFileException

  synchronized (Categories) {

    for (UAFile j : Categories.get(category).getAssociatedFiles().values())

      long stamp = lock(j.getFileName(), true);
      FileWriter writer = new FileWriter(j.getPathToFile());
      writer.append(contentToAppend);
      unlock(j.getFileName(), stamp, true);
      writer.close();
    }
  }

  return contentToAppend;
}

/**
 * implements a readLock on the file being put into the category as well as
 * synchronized access to the data structure "Categories" that we are adding
 *
 * @param file     file to be added to specified category
 * @param category category specified file will be added to
 * @return returns true/false based on if the file insertion was successful
 * @throws UAInvalidCategoryException
 * @throws UAInvalidFileException
 */

public boolean addFileToCategory(String file, String category)
    throws UAInvalidCategoryException, UAInvalidFileException {

  try {
    // lock the file
```

```
    File path = new File(file);
    UAFile fileToAdd = new UAFile(path);
    synchronized (Categories) {
      long stamp = lock(fileToAdd.getFileName(), false);

      if (Categories.get("nocategory").getAssociatedFiles().get(file) == nu

        synchronized (Files) {
          Files.put(fileToAdd.getFileName(), fileToAdd);
        }
        Categories.get(category).getAssociatedFiles().remove(fileToAdd.getFi

        path.createNewFile();
        fileToAdd.getAssociatedCategories().put(fileToAdd.getFileName(), nev
        Categories.get(category).getAssociatedFiles().put(fileToAdd.getFileN
      } else {

        Categories.get(category).getAssociatedFiles().remove(fileToAdd.getFi
        fileToAdd.getAssociatedCategories().put(fileToAdd.getFileName(), nev
        Categories.get(category).getAssociatedFiles().put(fileToAdd.getFileN
      }

      unlock(fileToAdd.getFileName(), stamp, false);
    }

    return true;
  } catch (Exception e) {

    return false;
  }
}

/**
 * removes a given file from a given category
 *
 * with synchronized Access to Categories and the file being removed
 *
 * @param file      file to be removed
 * @param category category the file will be removed from
 * @return returns true/false based on whether the removal was successful
 * @throws UAInvalidCategoryException
 */

public boolean removeFileFromCategory(String file, String category)
 throws UAInvalidCategoryException {
```

```
  try {
    long stamp = lock(Files.get(file).
    getFileName(), false);

    int listSize = Files.get(file)
    .getAssociatedCategories().size();

    synchronized (Categories) {

      if (listSize == 1 && Categories.get(category).getCategoryName().equals
        Categories.get("nocategory")
        .getAssociatedFiles().
        put(file, new UAFile(new File(file)));
      }
      if (Categories.get(category).getCategoryName().
      equals("nocategory")) {
        System.out.println("cannot remove category association to 'nocategor
        return false;
      } else {
        Categories.get(category).getAssociatedFiles().remove(file);
      }
    }

    unlock(Files.get(file).getFileName(), stamp, false);
    return true;
  } catch (Exception e) {
    return false;
  }
}

/**
 *
 * @param file given file to find the categories for
 * @return returns the categories a specified file is associated with in an
 *         HashMap
 * @throws UAInvalidFileException
 */

public HashMap<String, UACategory> getCategories(String file)
throws UAInvalidFileException {

  try {
    HashMap<String, UACategory> list = new HashMap<>();
    synchronized (Files) {

      for (UACategory j : Files.get(file).
```

```
      getAssociatedCategories().values()) {
        list.put(j.getCategoryName(), j);
        System.out.println(j.getCategoryName());
      }

    }

    return list;
  } catch (Exception e) {
    return null;
  }
}

/**
 *
 * @param category lists the associated files contained in a category
 * @return returns the associated files contained in the given category in a
 *         HashMap
 * @throws UAInvalidCategoryException
 */

public HashMap<String, UAFile> listFilesByCategory(String category)
throws UAInvalidCategoryException {

  try {
    HashMap<String, UAFile> list = new HashMap<>();
    synchronized (Categories) {

      for (UAFile j : Categories.get(category).
      getAssociatedFiles().values()) {
        long stamp = lock(j.getFileName(),false);

        list.put(j.getFileName(), j);
        System.out.println(j.getFileName());

        unlock(j.getFileName(),stamp,false);
      }

    }
    return list;
  } catch (Exception e) {
    return null;
  }
}

/**
```

```
 *
 * @param category given name of category to be created
 * @return returns the Object of the new category created
 */

public UACategory createCategory(String category)
throws UACategoryExistsException {

  UACategory c = new UACategory(category);
  this.lock.lock();
  if (lock.isHeldByCurrentThread()) {
    Categories.put(category, c);
  }
  this.lock.unlock();

  return c;

}

/**
 *
 * @param category given category to be deleted
 * @return returns the Object of the category deleted
 * @throws UACategoryExistsException
 * @throws UANotEmptyException
 */

public UACategory deleteCategory(String category)
throws UACategoryExistsException, UANotEmptyException {

  synchronized (Categories) {
    return Categories.remove(category);
  }

}

/**
 *
 * @param oldName previous name of category we want to change
 * @param newName name to replace the old category name
 * @return returns the new renamed category
 * @throws UACategoryExistsException
 * @throws UAInvalidCategoryException
 */

public UACategory editCategory(String oldName, String newName)
```

```
    throws UACategoryExistsException, UAInvalidCategoryException {

  UACategory temp = Categories.get(oldName);
  synchronized (Categories) {
    Categories.remove(oldName);
    Categories.put(newName, temp);
  }
  return Categories.get(newName);

}

/**
 *
 * @param list a given list of files to be index
 * @param i    i for recursion purposes
 */

public void index(File[] list, int i) {

  if (i >= list.length) {
    return;
  } else {
    if (list[i].isDirectory()) {
      index(list[i].listFiles(), 0);
    } else {
      UAFile file = new UAFile();
      file.setFileName(list[i].getName());
      file.setPathToFile(list[i].getAbsolutePath());
      Files.put(file.getFileName(), file);
    }
    index(list, ++i);
  }

}

/**
 *
 */

public void listCategories() {

  synchronized (Categories) {

    for (UACategory i : Categories.values()) {
      System.out.println(i.getCategoryName());
    }
```

```
  }
}

/**
 *
 * @return returns the list that it just printed in ascending order, unalter
 *         as I was not sure if you wanted us to alter the list or just pri
 */

public HashMap<String, UAFile> listAll() {

  String[] x = new String[Files.size()];
  int num = 0;
  for (UAFile i : Files.values()) {
    x[num] = Files.get(i.getFileName()).getFileName();
    num++;
  }
  mergeSort(x, 0, x.length - 1);
  for (int i = 0; i < x.length; i++) {
    System.out.println(x[i]);
  }

  return Files;
}

/**
 *
 * @param A Array to be sorted
 * @param p beginning index of array
 * @param r ending index of array
 */

public static void mergeSort(String[] A, int p, int r) {

  if (p < r) {
    int q = (p + r) / 2;
    mergeSort(A, p, q);
    mergeSort(A, q + 1, r);
    merge(A, p, q, r);
  }
}

/**
 *
 * @param A Array to merge
```

```
     * @param p beginning index of array
     * @param q middle index of array
     * @param r ending index of array
     */

    public static void merge(String[] A, int p, int q, int r) {

      int n1 = q - p + 1;
      int n2 = r - q;

      String[] L = new String[n1 + 1];
      String[] R = new String[n2 + 1];

      for (int i = 0; i < n1; i++) {
        L[i] = A[p + i];
      }
      for (int j = 0; j < n2; j++) {
        R[j] = A[q + j + 1];
      }

      L[n1] = "zzz";
      R[n2] = "zzz";

      int i = 0, j = 0;

      for (int k = p; k <= r; k++) {

        if (L[i].compareTo(R[j]) < 0) {
          A[k] = L[i];
          i++;
        } else {
          A[k] = R[j];
          j++;
        }
      }
    }

    @Override
    public void run() {

    }

}
```

UAFile Class

```
import java.io.File;
import java.util.HashMap;
import java.util.concurrent.locks.StampedLock;

/**
 * Each type of UAFile maintains fileName, the name of the file, pathToFile,
 * associatedCategories, which is HashMap of type <String, UACategory> of ass
 * this file, essentially which categories is this file a part of
 * @author noah_
 *
 */


public class UAFile {

  private String fileName;
  private String pathToFile;
  public StampedLock lock = new StampedLock();

  private HashMap<String,UACategory> associatedCategories = new HashMap<>();

  public String getFileName() {
    return fileName;
  }
  public void setFileName(String fileName) {
    this.fileName = fileName;
  }
  public String getPathToFile() {
    return pathToFile;
  }
  public void setPathToFile(String pathToFile) {
    this.pathToFile = pathToFile;
  }

  public UAFile() {

  }

  public UAFile(File file) {
    this.fileName = file.getName();
    this.pathToFile = file.getAbsolutePath();
  }

  public HashMap<String,UACategory> getAssociatedCategories(){
```

```
    return associatedCategories;
  }

}
```

UACategory Class

```
import java.util.HashMap;

/**
 * Contains the fields category name to maintain order of categories and a Has
 * of all associated files with this category, or better worded, files CONTAIN
 * @author noah_
 *
 */


public class UACategory {

  private String categoryName;

  private HashMap<String,UAFile> associatedFiles = new HashMap<>();

  public String getCategoryName() {
    return categoryName;
  }

  public void setCategoryName(String categoryName) {
    this.categoryName = categoryName;
  }

  public UACategory() {

  }

  public UACategory(String name) {
    this.categoryName = name;
  }

  public HashMap<String,UAFile> getAssociatedFiles(){
    return associatedFiles;
  }

}
```

UACategoryExistsException Class

```
public class UACategoryExistsException extends Exception {

  /**
   *
   */
  private static final long serialVersionUID = 1L;

  public UACategoryExistsException() {
    super("Category does not exist");
  }
}
```

UAInvalidCategoryException Class

```
public class UAInvalidCategoryException extends Exception {

  /**
   *
   */
  private static final long serialVersionUID = 1L;

  public UAInvalidCategoryException() {
    super("Invalid Category");
  }

}
```

UAInvalidFileException Class

```
public class UAInvalidFileException extends Exception {


  /**
   *
   */
  private static final long serialVersionUID = 1L;

  public UAInvalidFileException(){
    super("Invalid File");
```

```
  }
}
```

UANotEmptyException Class

```
public class UANotEmptyException extends Exception {

  /**
   *
   */
  private static final long serialVersionUID = 1L;

  public UANotEmptyException() {
    super("Category not empty");
  }
}
```